REMARKS

Claims 1-39 are pending in the present application. By this Response, claims 1, 7, 8, 12, 14, 20, 21, 25, 26, 27, 33, 34, and 38 are amended. Claim 26 is amended to reflect proper dependency on claim 14. Claims 1, 7, 8, 12, 14, 20, 21, 25, 27, 33, 34, and 38 are amended to provide proper antecedent basis for the "first object-oriented software package" and the "second object-oriented software package." No new matter is added as a result of the above amendments. Reconsideration of the above amendment to claims in view of the following Remarks is respectfully requested.

I. Objection to Claim 26

Claim 26 is objected to as being a duplicate of claim 13. By this Response, claim 26 is amended to depend on claim 14. Accordingly, Applicant respectfully requests the objection to claim 26 be withdrawn.

II. Telephone Interview Summary

Applicant thanks Examiner Romano and Supervisor Chen for extending the courtesies to Applicant's representative during the June 1, 2005 telephone interview. During the interview, Examiner Romano and Supervisor Chen agreed with the Applicant that Krishna fails to teach or suggest the features of identifying all public classes included in the first software object-oriented software package, for each of said public classes, identifying all public entities included in each of said public classes, or removing all references to software that is defined in a second object-oriented software package from said public entities included in each of said public classes, as alleged in the Office Action. Examiner Romano and Supervisor Chen also indicated that further search and consideration is necessary. The substance of the interview is summarized in the following remarks.

III. **35 U.S.C. § 103(a), Alleged Obviousness, Claims 1, 2, 4-7, 9-15, 17-20, 22-28, 30-33 and 35-39**

Claims 1, 2, 4-7, 9-15, 17-20, 22-28, 30-33 and 35-39 are rejected under 35 U.S.C. § 103(a) as being allegedly obvious over Krishna et al. (U.S. Patent Publication No. 2003/0051233) (hereinafter "Krishna") in view of Dale Green, "Trail: The Reflection API", The Java Tutorial posted November 27th, 1999 (hereinafter "Green"). This rejection is respectfully traversed.

Regarding independent claim 1, the Office Action states:

In regard to claim 1, **Krishna** discloses:

- *"A method in a data processing system for generating a generic compilation interface from a first object-oriented software package, said method comprising the steps of..."* (E.g., see Figure 7A & Page 2, Paragraph [0025]), wherein the library stubs exclude the source code executable statements, but include (generic), declarations and interfaces so that the secondary developer can compile class files for converting to CAP files, etc (interface).
- *"...removing all references to software that is defined in a second software package from said public entities included in each of said public classes..."* (E.g., see Figure 7B & Page 2, Paragraph [0025]), wherein the library stubs exclude (remove), the source code executable statements (software defined in a second package), but include declarations and interfaces so that the secondary developer can compile class files, wherein Figure 7B, steps 721- 734, teach replacing a reference returned with the appropriate return type value.
- *"... generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second package..."* (E.g., see Figure 7A & 7B, steps 737-747 & Page 3, Paragraph [0036]), wherein equivalent public class for each of said identified public class are generated, wherein "... only non-private (public) method signatures and field signatures are needed for off-card compiling and conversion.., is sufficient for synthesizing the library stubs 220 (Figure 3)".
- *"...compiling each of said equivalent public classes; and generating a compilation interface for said first package including each of said compiled equivalent public classes."* (E.g., see Figure 7A & 7B & Page 3, Paragraph [0048]), wherein the pseudo code teaches generating (compiling) an equivalent JAR file (Step 747).

But Krishna does not expressly disclose "*...identifying all public classes included in said first software package ...*" or "*...for each of said public classes, identifying all public entities included in each of said public classes ...*". However, Green discloses:

- "*... identifying all public classes included in said first software package* ."(E.g., see "Discovering Class Modifiers", Page 7), wherein all public class modifiers are discovered.
- - "*... for each of said public classes, identifying all public entities included in each of said public classes ...*" (E.g., see "Trail: The Reflection API", Page 1, Paragraph 1), wherein all public class modifiers are discovered along with their fields, methods and variables (entities).

Krishna and **Green** are analogous art because they are both concerned with the same field of endeavor, namely, using the JAVA language to examine, manipulate and work with classes. Therefore, at the time the invention was made, it would have been obvious to a person of ordinary skill in the art to combine a public class modifiers and their attributes with **Krishna's** JAVA program for interpreting, interfacing and compiling. The motivation to do so, is suggested by **Krishna**, "... only non-private method signatures and field signatures are needed... for compiling...", (Page 3, Paragraph [0035]), Furthermore, **Green** suggests "...to use the reflection API if you are writing development tools..." (Page 1, Paragraph 1).

Office Action dated February 24, 2005, pages 3-5.

Independent claim 1, which is representative of independent claims 14 and 27 with regard to similarly recited subject matter, recites:

1. A method in a data processing system for receiving a first object-oriented software package and using it to generate a generic compilation interface for use in an integrated development environment, said method comprising the steps of:

identifying all public classes included in said first object-oriented software package;

for each of said public classes, identifying all public entities included in each of said public classes;

removing all references to software that is defined in a second object-oriented software package from said public entities included in each of said public classes;

generating an equivalent public class for each of said identified public classes, said equivalent public class including equivalent public entities that include no references to said software defined in said second object-oriented software package;

compiling each of said equivalent public classes; and

generating a compilation interface for said first object-oriented

software package including each of said compiled equivalent public classes. (Emphasis added).

Neither Krishna nor Green teaches or suggests the features emphasized above. As discussed in the abstract, Krishna teaches a first software program that is being referenced by a second software program. A converter generates information including an interface definition for the first program in order to compile the second program. Also, a generator is provided for generating source code of the first program from the interface definition, so that source code of the second program may be developed and compiled from the first and second source code.

The Office Action alleges that Krishna teaches the features of identifying all public classes included in a first object-oriented software package and identifying all public entities included in each of the public classes in **Figure 7A**, which is shown below, and on page 2, paragraph 25, where Krishna teaches manually deriving a set of library stubs from Java Card library source code for secondary developers. The library stub excludes source code executable statements but includes declarations and interfaces of source code, so that secondary developers can compile class files for converting to converted applet files. **Figure 7A** of Krishna is shown below:

```
701  Parse arguments {
702  IDE file path = Set of directories indicating location of IDE files
703  jar file name = Name of jar file to be generated (containing
     synthesized classes)
704  IDE file name [optional] = IDE file from which to synthesize
     classes/jar file
705  }
706
707  For each IDE file to be parsed {
708  Parse IDE file and extract package name
709  For each class that belongs to the IDE file {
710  Initialize import list
711  Set the super class of the class in accordance with the super class
     info in the IDE file
712  For each field belonging to the class whose access_flag does not
     set ACC_INHERITED {
713  Create the field with the right signature and access condition
714  Process attributes {
715  If the field has a ConstantValue attribute {
716  Read the constant value and store it in the field
717  }
718  Update import list if field type refers to a class not in this package
719  }
```

FIG. 7A

In paragraph 25, Krishna teaches excluding or removing statements that are within the manually derived library stub. However, Krishna does not mention anything about identifying public classes within the source code or public entities from each identified public class. Krishna merely removes all executable statements, such as methods, from the library stubs leaving only the interfaces and declarations in the source code. There is no identification made by the manually derived library stubs of any public classes, let alone identification of public entities within the stubs. Therefore, Krishna does not teach identifying all public classes included in a first object-oriented software package and identifying all public entities included in each of the public classes, as recited in claims 1, 14, and 27 of the present invention.

In **Figure 7A**, Krishna teaches a pseudo code of the stub generator, which parses an IDE file and extracts its package name. Then, for each class in the file, the import list is initialized and a super class is set using information from the file. If any field of the class currently being processed is inherited from the super class (not overridden), a field is created with the right signature and an access

condition. If the field has a constant value attribute, the constant value is read and stored in the field. The import list is then updated for any field type that refers to a class not in this package (paragraph 28). Thus, Krishna merely teaches processing all classes within the IDE file and all fields that are not overridden or inherited from the super class. Krishna does not teach identifying all public classes that are included in the IDE file or all public entities that are in each of the public classes.

While, in paragraph 36, Krishna teaches that only non-private method signatures and field signatures are needed for off-card compiling and conversion, non-private method and field signatures are not the same as public classes or public entities within each public class as recited in claims 1, 14, and 27 of the present invention. As one of ordinary skill in the art would recognize, a class, method, or a field may be public, private, or protected. A non-private entity does not refer to a public entity, but rather refers to either a public or protected entity. Therefore, even though Krishna teaches non-private method and field signatures in the IDE file, it does not mean that Krishna teaches or suggests identifying all public classes in the IDE file or all public entities within each public class. To the contrary, as illustrated in **Figure 7A**, Krishna teaches processing all classes in the IDE file and any field that is not overridden. Krishna does not distinguish whether the classes and the fields are public, private, or protected. Therefore, Krishna does not teach the features of claims 1, 14, and 27 of the present invention.

In addition, the Office Action alleges that Krishna teaches removing all references to software that is defined in a second object-oriented software package from the public entities included in each of the public classes in **Figure 7B** and on page 2, paragraph 25, where Krishna teaches excluding or removing executable statements that are within the manually derived library stub. **Figure 7B** of Krishna is shown below:

```
720  For each method belonging to the class whose access_flag does not set
     ACC_INHERITED {
721  Create the method header with the right signature and access condition
722  If method name begins with <init> (a constructor method) {
723  Set name to correspond to the class name
724  Set return type to null
725  }
726  If the method returns a reference
727  Set the return value to null
728  Else
729  Set the return value to 0 cast appropriately to match the method's return type
730  Process attributes {
731  If the method has an Exceptions attribute {
732  Read the fields attribute and Store the throws clause(s)
733  }
734  }
735  Update import list if method return or argument type refers to class not
     in this package
736  }
737  Synthesize class file {
738  Create a source file named <classname>.java in the appropriate directory
739  Append package statement
740  Append the imports list
741  Append the class/interface statement with superclass/superinterface list
742  Append field declarations
743  Append method stubs
744  Compile the generated source
745  }
746  }
747  Place the synthesized class files in a .tar file
748  }
```

FIG. 7B

In paragraph 25, Krishna teaches removing executable statements that are within the library stub and leaving only interfaces and declarations. However, Krishna fails to mention anything about removing references to software that is defined in a second object-oriented software package. To the contrary, Krishna teaches away from removing references to software that is defined in a second object-oriented software package by specifically teaching in line 718 of **Figure 7A** that the import list is updated if the field type refers to a class that is not the package. Thus, Krishna teaches updating the import list to include references to software that is defined in a second object-oriented software package, not to remove references to software that is defined in the second object-oriented software package.

Additionally, in **Figure 7B**, the stub generator processes any method that is not overridden by creating a method header with the right signature and access condition,

setting the method name to correspond to the class name if the method is a constructor method, setting the return type as the return value if the method returns a reference, and creating information from exception attribute if the method has exception attribute. The Examiner alleges that replacing a reference returned with the appropriate return type value is the same as removing reference to software defined in the second object-oriented software package (page 4, paragraphs 48-49). Applicant respectfully disagrees.

In line 726-729 of **Figure 7B**, Krishna determines if the method returns a reference. If so, the return value is set to return a null object. Otherwise, the return value is set to 0 and the return value is cast to match the method's return type. Thus, if the return type of the method is an integer, the return value of 0 is cast to an integer. However, the return value is merely a return object, which is either a null object or an integer object. The return value is not a reference to software defined in a second software object-oriented software package.

To the contrary, Krishna again teaches away from removing references to software defined in a second software by specifically teaching, in line **735 of Figure 7B**, that the import list is updated to refer to classes that are not in the software package if the method return or if the argument type refers to classes that are not in the software package. Thus, instead of removing the reference to software that is outside the software package, Krishna updates the import list to include a reference to software that is outside the package in the import list. Therefore, Krishna does not teach or suggest removing all references to software that is defined in a second object-oriented software package from the public entities included in each of the public classes as recited in claims 1, 14, and 27 of the present invention.

Furthermore, Krishna does not teach generating an equivalent public class for each of the identified public classes, the equivalent public class including equivalent public entities that include no references to the software defined in the second object-oriented software package. The Office Action alleges that Krishna teaches these features in **Figure 7A** and **7B**, lines **737-747** and on page 3, paragraph 36, where Krishna teaches only non-private methods signatures and field signatures are needed for off-card compiling and conversion. As discussed above, non-private method and field signatures are not the same as public classes

or public entities within each public class. Since Krishna fails to mention anything about identifying all public classes or public entities within each public class, Krishan would not teach generating an equivalent public class for each identified public class that includes equivalent public entities that include no reference to software that is defined in the second object-oriented software package. To the contrary, as discussed above, Krishna teaches updating the import list to refer to classes that are not in the same software package.

Moreover, in lines 737-747 of **Figure 7B**, Krishna teaches creating a source file by appending a package statement, an imports list, a class/interface statement with superclass/superinterface lists, field declarations, and method stubs. Thus, Krishna teaches creating a source file that includes an imports list that refers to classes that are not in the same software package. Krishna does not generate an equivalent public class including equivalent public entities that include no references to the software defined in the second object-oriented software package. Contrary to not including references to software that is defined in the second object-oriented software package, Krishna specifically includes references to software that is defined in a second object-oriented software package. Therefore, Krishna does not teach the features of claims 1, 14, and 27 of the present invention.

Green also does not teach the features of claims 1, 14, and 27 of the present invention. On page 1, paragraph 1, Green teaches a Reflection API that represents classes, interfaces, and objects in the current Java Virtual Machine. By using the API, a user may determine the class of an object, get information about the class's modifiers, fields, methods, constructors, and superclasses, find out what constants and method declarations belong to the interface, create an instance of a class whose name is unknown until runtime, get and set value of the an object's field even if the field name is unknown until runtime, invoke a method on an object that is unknown until runtime and create a new array, whose size and component type are not known until runtime and modify the array's components.

While Green provides the capability of invoking a method on an object that is unknown until runtime, Green does not teach or suggest removing all

references to software that is defined in a second object-oriented software package. As Green merely invokes a method of a class, Green does not remove references that refer to a second object-oriented software package from the class. Therefore, Green also does not teach the feature of claims 1, 14, and 27 of the present invention.

Furthermore, it would not have been obvious for a person of ordinary skill in the art to modify or combine the teachings of Krishna and Green to reach the presently claimed invention. Krishna is concerned with generating a library stub that includes an import list, which refers to software that is defined in a second object-oriented software package. Green is concerned with providing a Reflection API for determining the class modifiers, fields, methods, constructors, and superclasses. Neither Krishna nor Green teaches or suggests removing all references to software that is defined in a second object-oriented software package from public entities that are identified in each of the public classes. Even if a person of ordinary skill in the art were to combine the teachings of Krishna and Green, the resulting combination would not be removing all references to software that is defined in a second object-oriented software package from public entities that are identified in each of the public classes. Rather, the resulting combination would be including all references to software that is defined in a second object-oriented software package from public entities that are identified using the Reflections API. Therefore, a person of ordinary skill in the art would not have been led to either modify or combine the teachings of Krishna and Green.

In view of the above, Applicant respectfully submits that neither Krishna nor Green teaches or suggests the features of claims 1, 14, and 27. At least by virtue of their dependency on claims 1, 14, and 27 respectively, neither Krishna nor Green teaches or suggests the features of dependent claims 2, 4-7, 9-13, 15, 17-20, 22-26, 28, 30-33, and 35-39. Accordingly, Applicant respectfully requests withdrawal of the rejection of claims 1, 2, 4-7, 9-15, 17-20, 22-28, 30-33 and 35-39 under 35 U.S.C. § 103(a).

IV. 35 U.S.C. § 103(a), Alleged Obviousness, Claims 3, 16, and 29

Claims 3, 16, and 29 are rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Krishna in view of Green and further in view of Evans et al. (U.S. Patent No. 6,836,884) (hereinafter "Evans"). This rejection is respectfully traversed.

Dependent claim 3, which is representative of claims 16 and 29, recites:

> 3. The method according to claim 1, further comprising the steps of:
> determining whether each of said entities includes a native attribute;
> in response to a determination that each of said entities includes a native attribute, removing said native attribute from each of said entities; and
> generating equivalent entities that include no native attributes.
> (Emphasis added)

Krishna, Green, and Evans fail to teach the features emphasized above. As discussed in the Abstract, Evans teaches a method for editing software program in a common language runtime environment. The method suspends execution of a portion of the native code component and allows users to edit the source code component to create an edited source code component. The edited source code component is compiled to create an edited native code component, which is then executed at the point where the execution was previously suspended.

While the Office Action admits that neither Krishna nor Green teaches the features above, the Office Action alleges that Evans teaches these features in Figure 3 and at column 12, line 63 to column 13, line 11, which reads as follows:

> The edit and continue component **104** of the runtime system **100** may thus be employed to change an existing method of a class. For example, the user **112** may add a new variable to the existing method, and/or to change an algorithm or function in the existing method. In this regard, the edit and continue component **104** may substitute edited native code (e.g., edited native code component **174** of **FIG. 7**) corresponding to the existing method (e.g., method **1A**) upon a return to the method, which return may be determined via a breakpoint. In addition, the edit and continue component (e.g., as well as the debugger application **110**) may interact with more than one source code compiler (e.g., source compiler **118**), whereby the edit and continue component **104** allows the user **112** to replace an existing method create in a first source language with a new method created in a second source language (e.g., Visual Basic, C++, C#, Java Script, APL, COBOL, Pascal, Eiffel, Haskell, ML, Oberon, Perl,

> Python, Scheme, Smalltalk, Objective Caml, and the like). (Emphasis added).

(Column 12, line 63 to column 13, line 11, Evans).

In the above section, Evans teaches editing an existing method of a class by substituting edited native code corresponding to the existing method upon a return to the method. However, Evans does not teach in response to a determination that each of said entities includes a native attribute, removing said native attribute from each of said entities, and generating equivalent entities that include no native attributes. Evans merely teaches editing existing native code that corresponds to an existing method. Evans does not teach removing a native attribute from each entity that includes a native attribute and generating equivalent entities that include no native attributes. To the contrary, Evans teaches, at column 5, lines 37-55, that the edited source code component is compiled by a source code compiler to create edited intermediate language component, which is then compiled by an intermediate language compiler to create an edited native code component to be executed at the beginning of the point where the execution of the native code component was originally suspended.

Since the edited source code component is compiled into an edited native code component, the edited native code component would still include native attributes. This is different from the presently claimed invention, in which all native attributes are removed from public entities and equivalent entities are generated with no native attributes. Therefore, Evans teaches away from removing native attributes from public entities by specifically teaching creating edited native code component that still includes native attributes. Therefore, Evans does not teach the features of claims 3, 16, and 29 of the present invention.

In view of the above, Applicant respectfully submits that Krishna, Green, and Evans fail to teach or suggest the features of claims 3, 16, and 29. Accordingly, Applicant respectfully requests withdrawal of the rejection of claims 3, 16, and 29 under 35 U.S.C. § 103(a).

V. **35 U.S.C. § 103(a), Alleged Obviousness, Claims 8, 21, and 34**

Claims 8, 21, and 34 are rejected under 35 U.S.C. § 103(a) as being allegedly unpatentable over Krishna in view of Green and further in view of obviousness. This rejection is respectfully traversed.

As discussed above in arguments presented for claims 1, 14, and 27, neither Krishna nor Green teaches or suggests the features of claims 1, 14, and 27 from which claims 8, 21, and 34 depend. In addition, it would not have been obvious to a person of ordinary skill in the art to modify or combine the teachings of Krishna and Green to reach the presently claimed invention. Krishna is concerned with including references to software that is defined in a second object-oriented software package rather than removing the references. Green is concerned with providing a Reflections API to determine a class's modifiers, fields, methods, constructors, and superclasses. Neither Krishna nor Green teaches or suggests that reference to software that is defined in the second object-oriented software package to be removed. Therefore, it would not have been obvious to a person of ordinary skill in the art to modify or combine the teachings of Krishna and Green to reach the features of claims 1, 14, and 27, from which claims 8, 21, and 34 depend.

In view of the above, Applicant respectfully submits that neither Krishna nor Green teaches or suggests the features of claims 1, 14, and 27. At least by virtue of their dependency on claims 1, 14 and 27 respectively, neither Krishna nor Green teaches or suggests the features of dependent claims 8, 21, and 34. Accordingly, Applicant respectfully requests withdrawal of the rejection of claims 8, 21, and 34 under 35 U.S.C. § 103(a).

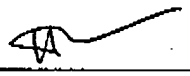
VI. Conclusion

It is respectfully urged that the subject application is patentable over the cited references and is now in condition for allowance.

The Examiner is invited to call the undersigned at the below-listed telephone number if in the opinion of the Examiner such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: June 14, 2005

Respectfully submitted,



Wing Yan Mok
Reg. No. 56,237
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Agent for Applicant